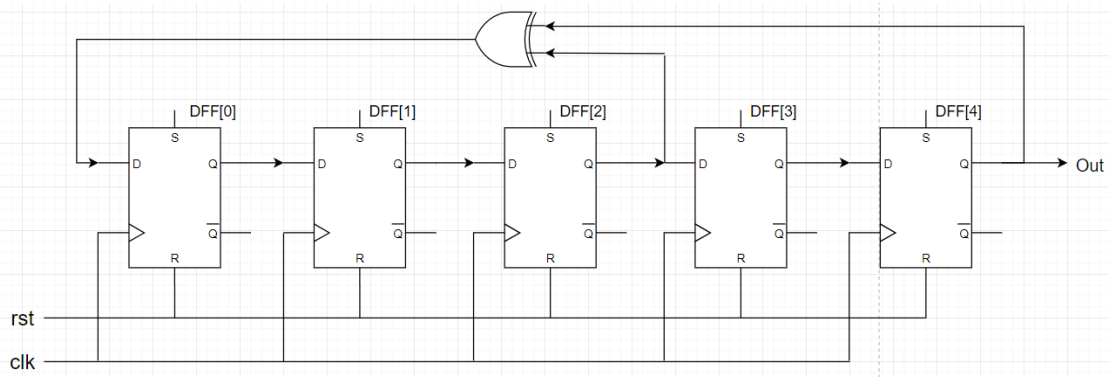
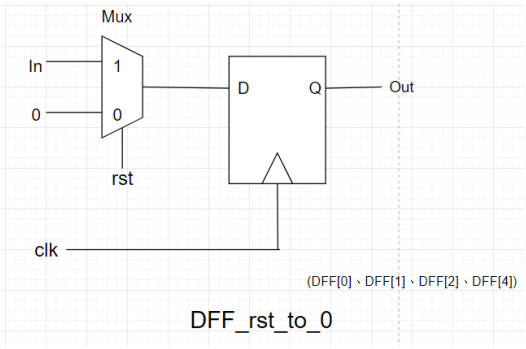
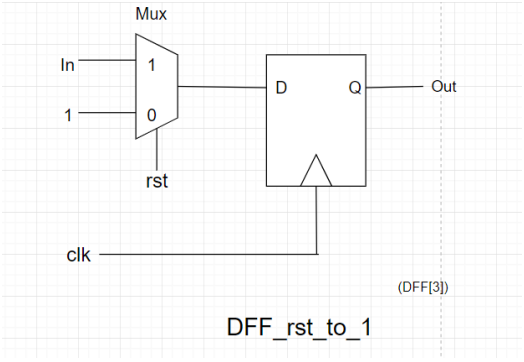
Question 1

由題幹本身所給的 Linear-Feedback Shift Register (LFSR)簡圖，加上 RESET 訊號後可以畫出以下的電路圖：

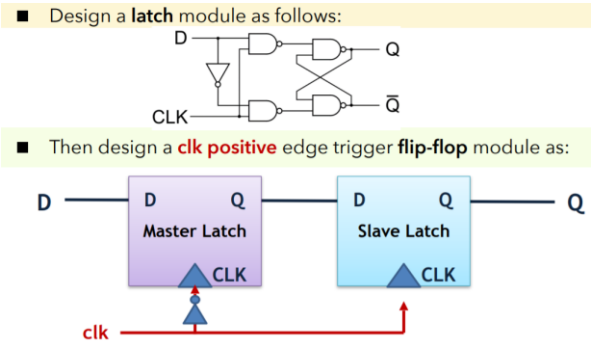


如題所需：When RESET == 1'b0, reset DFF[4:0] to 5'b01000，因此當 RESET == 1'b0 時，DFF[0]、DFF[1]、DFF[2]以及 DFF[4]要 RESET to 0；DFF[3]要 RESET to 1。個別的電路圖如下：



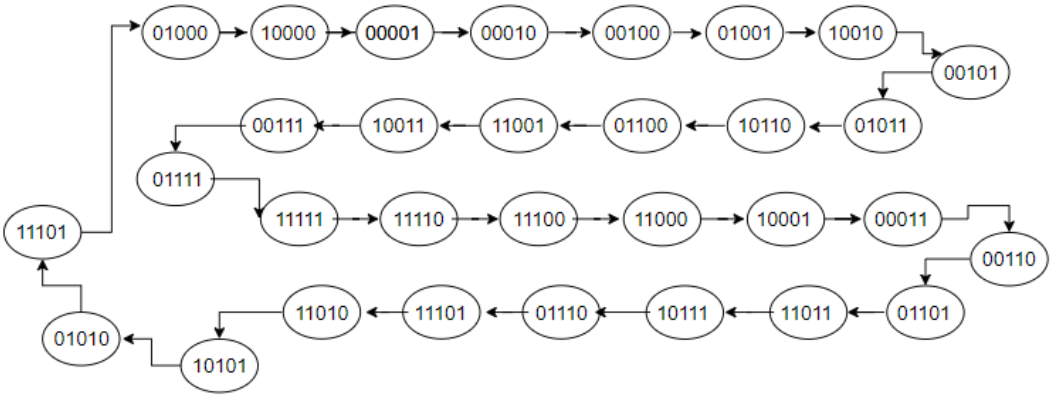


而 D-Filp-Flop(DFF)由 lab2 的 basic question 2 又可以知道更確切的構造，如下：



而由 LFSR 的設計構造，可以一步一步推算出每經過一個 clock positive edge

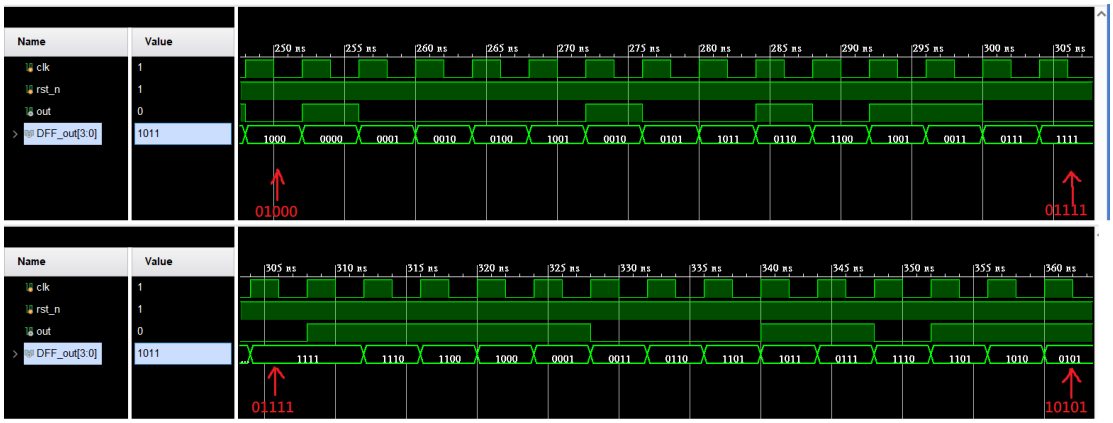
trigger 後各個 DFF 所存入的值，也就是可以推算出每一個 state，進而畫出 State Transition Diagram，如下圖所示：

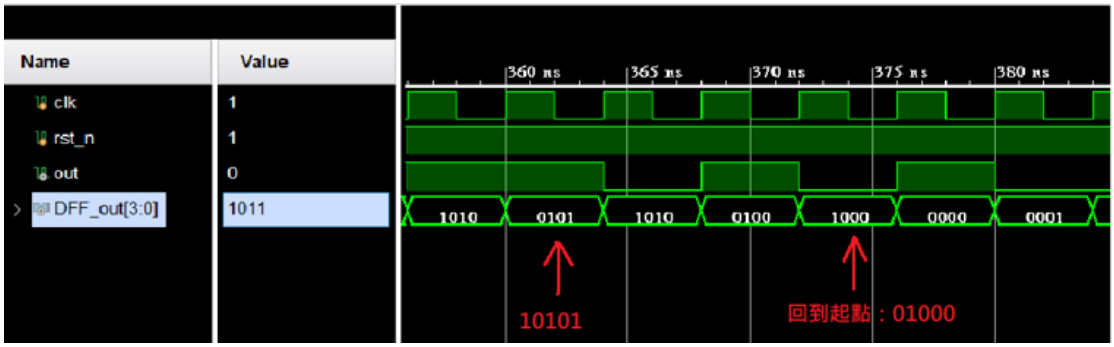


State Transition Diagram of DFFs

此 State Transition Diagram 共有 31 個 states，如果 DFF[4:0]以 5'b01000 為起點開始算起，過了 31 個 state 後又會回到自己本身的 state，依照著張圖的順序不斷輪轉。

由 Vivado 可模擬出以下的波形圖：(下圖以{out , DFF_out[3:0]}去看)





由上方的波形模擬可證實每經過 31 個 state 就會回到起點，並且不斷循環
以上 simulation results 皆符合 State Transition Diagram 預期。

Testbench 設計理念：

因為此題 Template 給的 output 只有最後一個輸出的 out(DFF[4])，為了檢測電路是否正確，我在 LFSR.v 中有多宣告 output [3:0] DFF，還有在 LFSR_t.v 中多宣告 wire [3:0] DFF(但繳交檔案到 ilms 時我會 comment 掉)，如上的 Vivado 波形模擬圖所顯示。

此題的檢測方式很簡單，一開始 rst_n = 1'b0，讓 Register(那五個 DFFs) reset to 5'b01000，接著就使 rst_n = 1'b1，讓 LFSR 正常運作，因為只有 31 個 states，所以當觀察到回到原本的起點時，就完成了電路檢測，因為經過下一個還是會經過一樣的循環。

When RESET == 1'b0, reset DFF[4:0] to 5'b00000. What will happen next?

Ans: 當 RESET == 1'b0, reset DFF[4:0] to 5'b00000. 而當下一個階段 RESET == 1'b0 時，每經過一個 clock trigger，DFF[4:0]就會開始往右移一位，明顯地，DFF[4:1] = 4'b0000，而 DFF[0] = DFF[4]⊕DFF[2] (DFF[4] xor DFF[2]) = 0，因此每經過一次，DFF[4:0]的值都不會變，此時就沒有移位的效果。

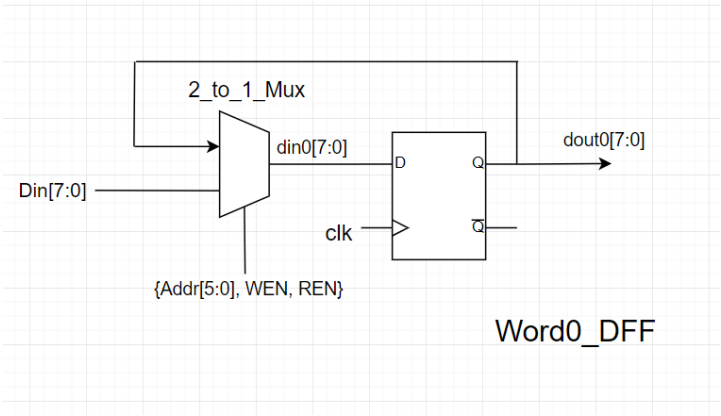
Question 2

Input : clk, REN, WEN, Addr[5:0], Din[7:0];
Output : Dout[7:0];

電路設計流程：

Specification1 : (When WEN == 1'b1, write Din to MEM[Addr])&&(If REN and WEN are both 1, do read only)

⇨ 因為要寫 Din[7:0]到指定的 Addr[5:0] (Word)，因為總共有 64 個 Words，所以要有 64 個 8-bit-DFFs 去存值。而 DFF 可否被存值的條件只有當(WEN == 1 && REN == 0 && Addr == 指定的 Word)，輸入的 Din 才會被存起來；當條件不符合時，就保持原來有的值。以上所述的條件，就等同於 DFF 前面還有一個 2_to_1 的 Mux(多工器)，Mux 選擇的控制條件為{WEN, REN, Addr}三者。綜合以上 **Spec** 的條件，可以畫出以下的電路圖：(下圖是畫出存取 Word0 值的 DFF)



而控制條件為{WEN, REN, Addr}三者我是用 if, else if 判斷式寫的，所以 2_to_1_Mux 控制訊號的詳細條件我用文字表達，邏輯圖也就沒畫出來了。

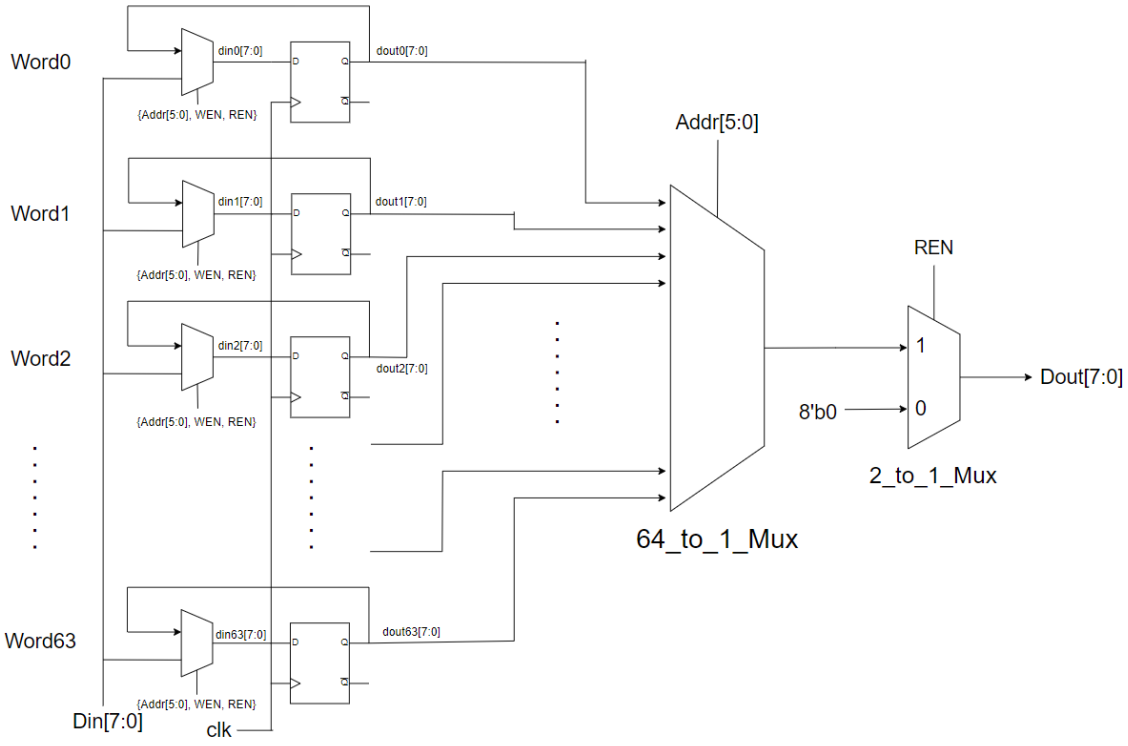
<補>判斷 Addr[5:0]是否等於指定的 Word，可以將 Addr[5:0]六個個別位數去跟指定 Word 六個個別位數的 address bit Xnor 起來，再全部 And 起來，如果 And 的結果是 1，代表是指定的 Word；反之則反。

Specification2 : When REN == 1'b1, output MEM[Addr] to Dout; otherwise Dout = 8'd0

⇨ 首先，因為有 64 個 8-bit DFFs 會出來，所以需要一個 64_to_1 的 Mux 選出要 output 哪一個 Word 的值，64_to_1 的 Mux 選擇的控制條件為 Word 的 address。最後將 Word 的 output 接到一個 2_to_1_Mux 的一個 input，另一個 input 接 8'd0，如果 REN == 1'b1，out = Word 的 output；如果

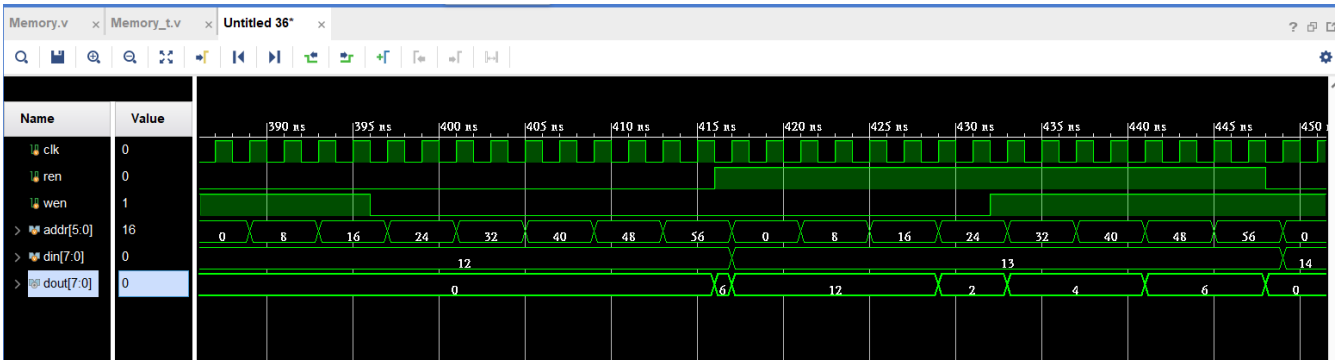
REN == 1'b0，out = 8'd0。

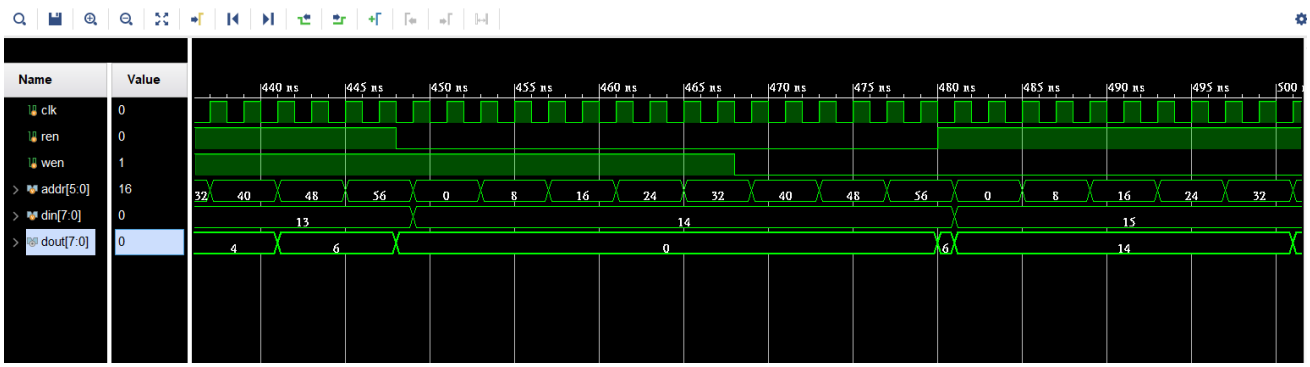
綜合以上 **Spec1,2**，可以畫出以下的電路圖：



64 8-bit DFFs in total

由 Vivado 可模擬出以下的波形圖：





由上方的波形模擬可證實以上的 case 皆符合 Specification，特別是當(REN == 1)&&(WEN == 1)時，只會做讀取的動作。以上 simulation results 皆符合預期。

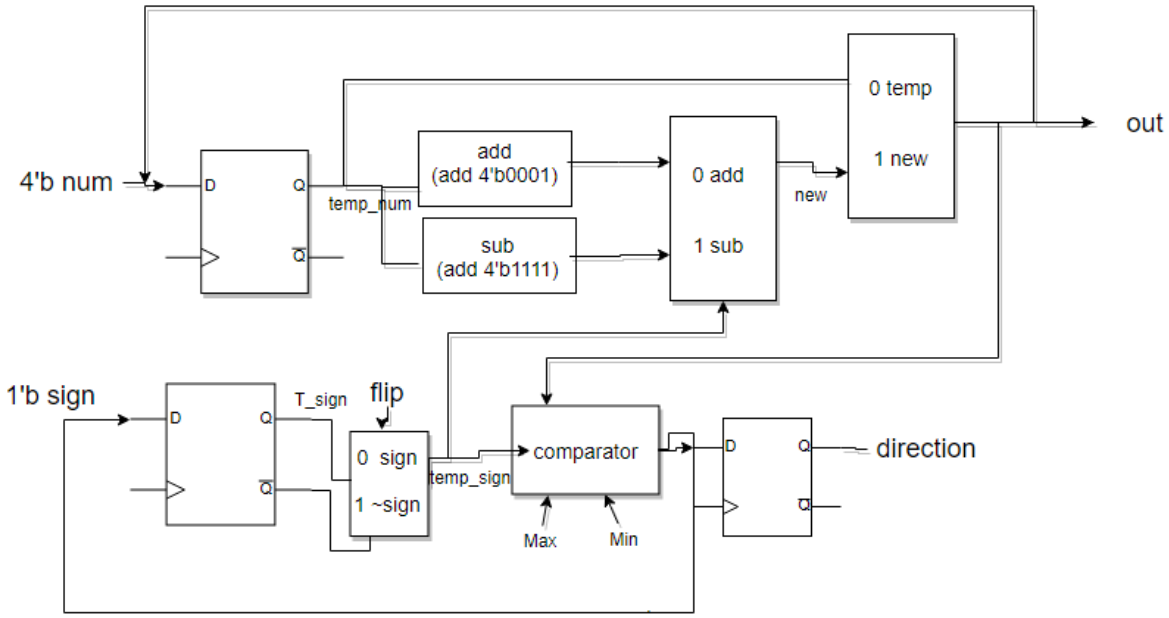
Testbench 設計理念：

在 testbench 中，一開始預設{din, addr} = 14'd0，預設 clk 每過 1 個時間單位震盪一次(always#1 clk = ~clk)、REN 每過 32 個時間單位震盪一次(always#32 REN = ~REN)、WEN 每過 36 個時間單位震盪一次(always#36 WEN = ~WEN)。這樣，在不同時間會有 4 種情況，像是(WEN==0&&REN==0)、(WEN==1&&REN==0)、(WEN==0&&REN==1)或是(WEN==1&&REN==1)。而我在每過 4 個時間單位{din, addr} ={din, addr} + 14'd8，所以代表我只抓其中幾個 Address 來看讀取 output 值(只看 Address 0, 8, 16, 24, 32, 40…..)，如果那幾個 Address 的讀寫皆是正常的，那其他的 Address 的讀寫基本上也會是正常的，因為 64 個 case 太多了，我直接用 C++ 打迴圈 print 所有變數，然後貼在 vivado 上，所以不會有接錯線的問題。

Question 3

實作方式 ：

實作之流程圖為以下所示。



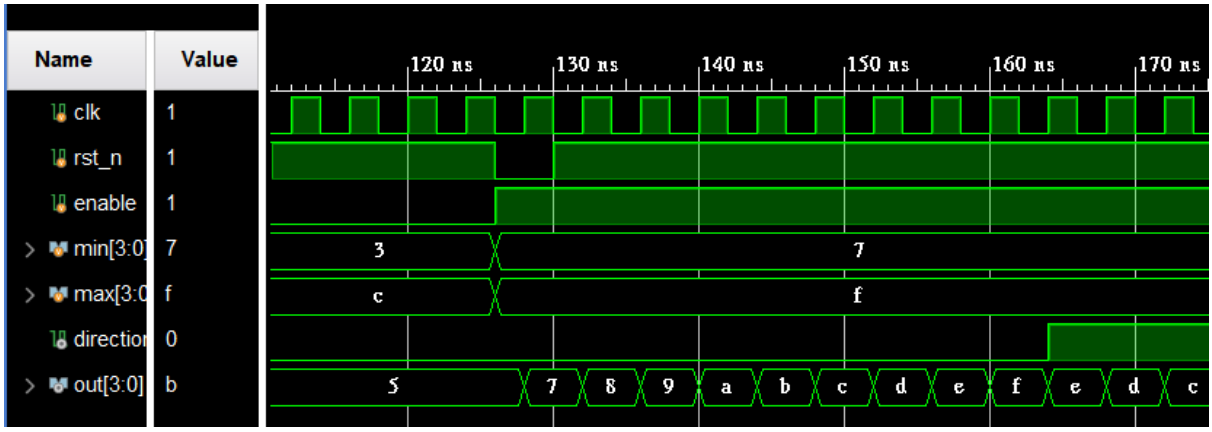
首先解釋 num，num 經過一個 clk 週期之後會存在 temp_num 的地方，同時會做一個加法運算及減法運算，經由 sign 來選擇，若是 sign 為 0，代表 counter 會向上數，因此選擇 add 的結果，sign 選擇完過後會存在 new 這個地方，此時要再經過一個多功器，若是 EN 訊號為一，代表 counter 可以運作，選擇 new 的值輸出，若為零，則選擇原訊號輸出。

再來解釋 sign 的運作方式，首先 sign 會先經過一個由 flip 選擇的多功器，若是 flip 為一，選擇反向輸出，若為零則正常輸出，再來則進入邊界檢查的環節，也就是經由 comparator 來比較，若是當時候的 out 已經為最大值或最小值時，sign 會變號，將下一個 clock 來臨時，最大值與最小值會做反向運算。

最後是 direction 的部分，由於 direction 會慢 sign 一個週期，因此會接一個 D-flip-flop 然後輸出。

Testbench 的設計&驗證 ：

verilog 實做出來之後，利用 testbench 跑 simulation 做確認，驗證我們打的程式是否正確。testbench 的設計方式是丟入各種較為特殊的測資，然後觀察這些 output 是否為我們想要的結果。

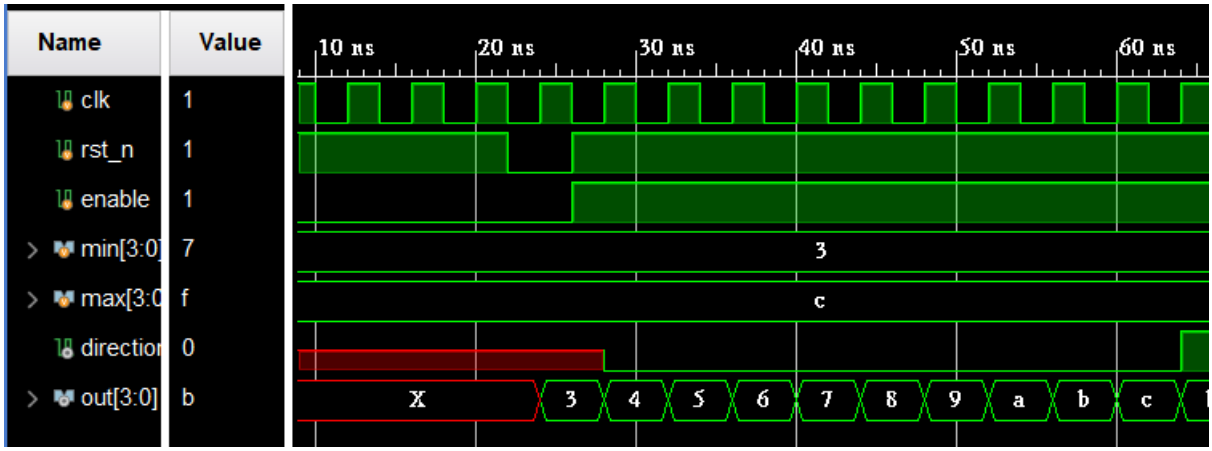


可觀察到 1).在經過 reset 之後，out 的值變為最小值 7

2).在 enable 等於 0 時，out 的值不變

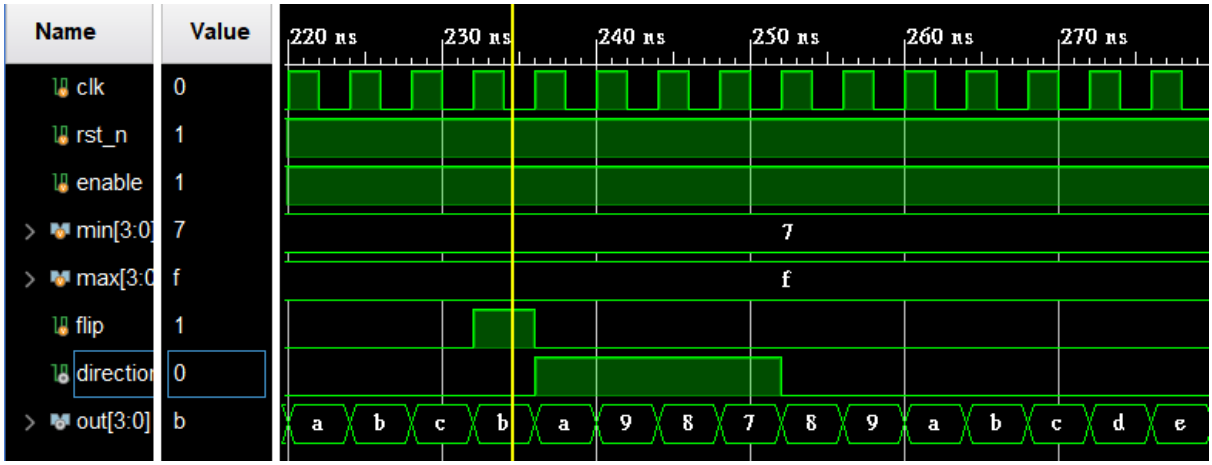
3).上途中的 direction 皆為正確

4). out 的值在 7 與 f 之間來回震盪

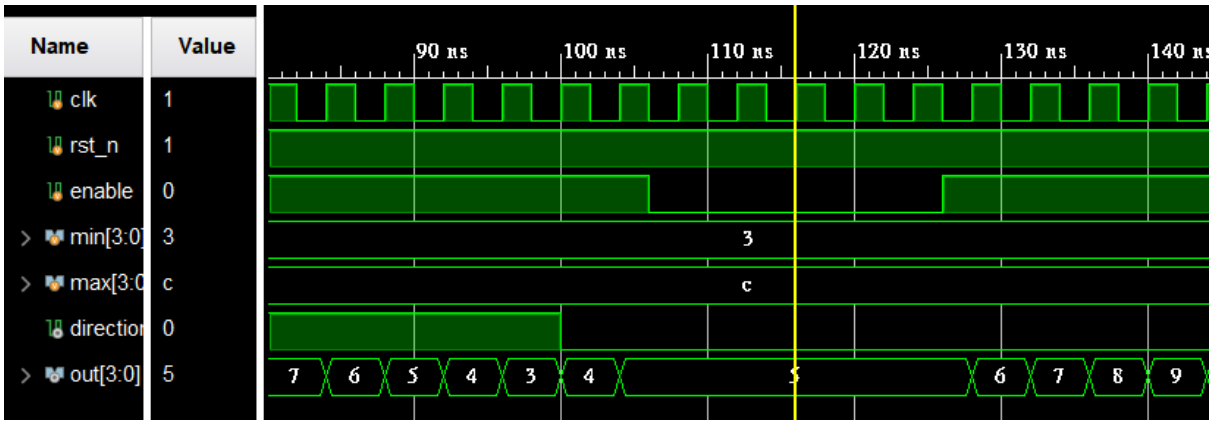


可觀察到 1).在經過 reset 之後，out 的值變為最小值 3

2).上途中的 direction 皆為正確

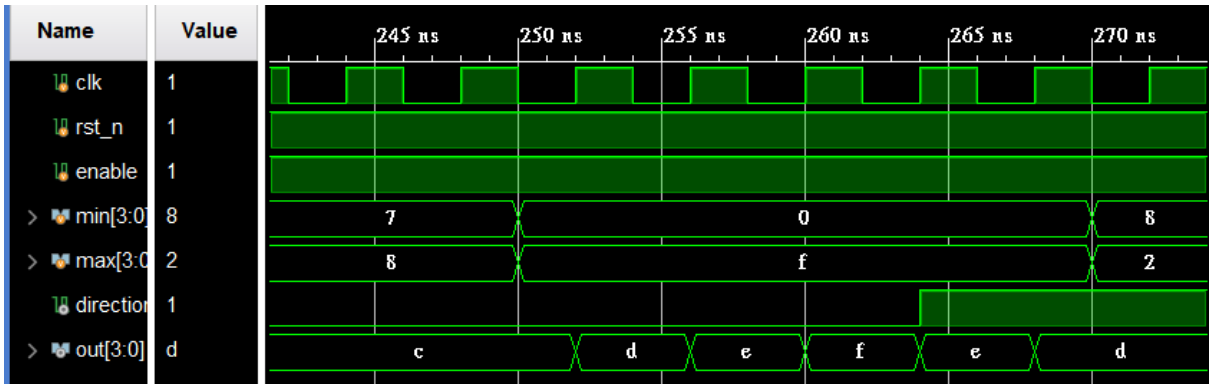


可觀察到 1)當 flip 訊號來時，direction 方向會改變



可觀察到 1).在 enable 等於 0 時，out 的值不變

2). 在將值 hold 住時，dierection 方向仍正確



可觀察到 1). out 值不在 min 與 max 的範圍之內時，其值 hold 住

2). direction 方向仍一致

藉由以上的各個情況的結果，我們可以說這份設計是正確的

FPGA demonstration

接下來我們要將 verilog code 跑在板子上面，此分為兩個重要的環節。

(一) 顯示

(a). decoder : 為了將數字轉換為阿拉伯數字圖形，必須設計一個 decoder 當要表示一個數字時，會有對應的一組控制七段顯示器的訊號輸出。

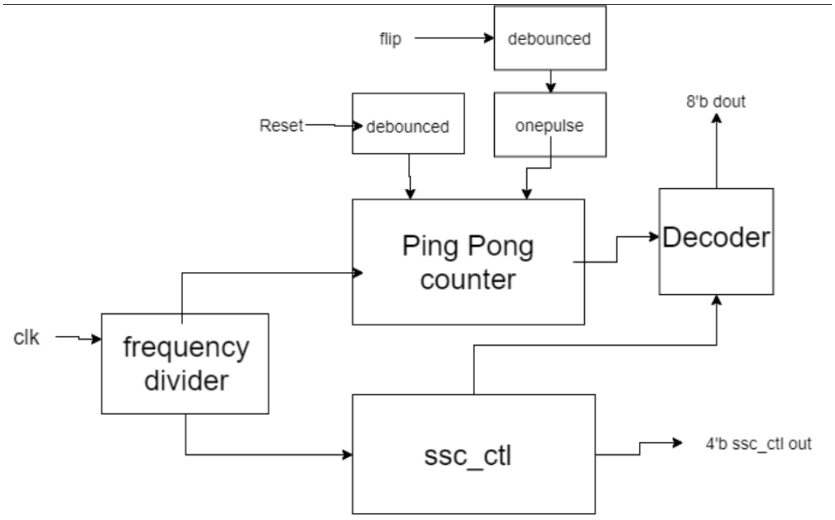
(b). 七段顯示器控制 : 為了讓每個七段顯示器輪流亮燈，用 2'b 的 EN 訊號控制，一個時間只會有一個七段顯示器亮。

(二) 訊號處理

(a). 除頻器 : 首先我們要將石英震盪器的頻率除頻至人眼可已看得到的頻率，因此我們先做一個 26'b 的加法器，最高位的那一項輸出作為 counter 的週期，此外我們也輸出了第 15 和第 16 個 bit 作為輪流顯示七段顯示器的控制訊號。

(b). debounce : 由於用按壓按鈕的初期，按鈕會震盪而產生雜訊，為了過濾訊號，接了若干組 flip flop，當每個 flip flop 的值皆為一時才輸出一。

(c). onepulse : 在 flip 作用的時候，我們只需要效果持續一個週期，因此在 debounce 完之後，會接一個反向器做 AND，讓 flip 的值為一的時間僅有一個週期以下位 FPGA 的設計方塊圖



心得

這次的 Lab 是正式接觸到 sequential circuit 的第一次，老實說花了不少時間才搞懂這些東西，也是到處去問過上學期修過邏設實驗的電機系同學，一步一腳印才成功把 Lab3 打出來。雖然辛苦，但是多了 clk 真的變得豐富很多，而且真的學到很多東西，心理是興奮且踏實的。感覺之後的 Lab 也會越來越有趣(難)，尤其 FPGA demo 這塊，希望我的分數能夠堅持住，因為真的花了很多時間在這上面。還有，真的可以問資工系會長大大嗎？希望助教或教授能夠幫幫我，在課堂上下個聖旨之類的！XDDD (107060011 涂皓鈞)

這次做 counter 及 fpga 實驗花了無數的時間與心力，第一次打程式打到天亮，從晚上十二點打到中午十二點，雖然過程中是痛苦，不過也讓我邏輯設計的改念更清楚，verilog 語言也漸漸能夠掌握了，打完獲得了滿滿的成就感。此次實驗也學習到了審查題目的重要性，像這次實驗沒有注意到 clk 的限制，因此浪費了很多時間在改程式，下次會更加謹慎。(107060015 陳弘軽)

小組分工

涂皓鈞 – 負責 Question 1、2 的 verilog 以及對應題號的結報內容 + 心得
陳弘軽 – 負責 Question3、FPGA demonstration 的 verilog 以及對應題號的結報內容 + 心得